# *SBC-ComEx User's Manual*

# *SBC-ComEx User's Manual*

The SBC-ComEx User's Manual was prepared by the technical staff of Innovative Integration on August 7, 2008.

For further assistance contact:


Innovative Integration
2390-A Ward Ave
Simi Valley, California 93065

PH: (805) 578-4260
FAX: (805) 578-4225

email: techsprt@innovative-dsp.com
Website: www.innovative-dsp.com


This document is copyright 2008 by Innovative Integration. All rights are reserved.

VSS \ Distributions \ SBC-ComEx \ Documentation \ Manual \ SBC-ComExMaster.odm

#XXXXXX

Rev 1.0

# Table of Contents


**SBC-ComEx User's Manual....2**

**Introduction....7**

Real Time Solutions!....7
Vocabulary....7
What is Malibu? ....7
What is C++ Builder?....7
What is Microsoft MSVC?....8
What kinds of applications are possible with Innovative Integration hardware?....8
Why do I need to use Malibu with my Baseboard?....8
Finding detailed information on Malibu....8
Online Help....8
Innovative Integration Technical Support....9
Innovative Integration Web Site....9
Typographic Conventions....9

**Windows Installation....10**

Host Hardware Requirements....10
Software Installation....10
Tools Registration....13
Bus Master Memory Reservation Applet....13
Hardware Installation....14
After Power-up....14

**Installation on Linux....16**

Package File Names....16
Prerequisites for Installation....16
The Redistribution Package Group - MalibuRed....16
Malibu....17
Other Software....17
Baseboard Package Installation Procedure....17
Board Packages....18
Unpacking the Package....18
Creating Symbolic Links....18
Completing the Board Install....19
Linux Directory Structure....19
Applets....19
Documentation....19
Examples....19
Hardware....19

**Writing Custom Applications....20**

SBC-ComEx Example Software....20
Tools Required....20
Program Design....21
The Host Application ....21
User Interface....21

Configure Tab....21
Clock Tab....22
Trigger Tab....23
Host Side Program Organization....23
ApplicationIo....24
Initialization....24
Using the Programmable Timebase....26
Using Programmable Bit I/O....27

Polling Thread....27
GPS Support ....28

**Developing Host Applications....31**

Borland Turbo C++....31
Microsoft Visual Studio 2005....33

**Applets....35**

Common Applets....35
Registration Utility (NewUser.exe)....35
Reserve  Memory Applet (ReserveMemDsp.exe)....36
....36
Data Analysis Applets....36
Binary File Viewer Utility (BinView.exe)....36

**Carrier Hardware....37**

Introduction....37
Features....37
Supports COM Express CPU Modules....37
Dual XMC Sites....37
Host Card Peripherals....38
PCI Interface....38

# List of Tables

Table 1. Development Tools for the SBC-ComEx Example.....20

# List of Figures

# *Introduction*

## *Real Time Solutions!*

Thank you for choosing Innovative Integration, we appreciate your business! Since 1988, Innovative Integration has grown to become one of the world's leading suppliers of DSP and data acquisition solutions. Innovative offers a product portfolio unrivaled in its depth and its range of performance and I/O capabilities .

Whether you are seeking a simple DSP development platform or a complex, multiprocessor, multichannel data acquisition system, Innovative Integration has the solution. To enhance your productivity, our hardware products are supported by comprehensive software libraries and device drivers providing optimal performance and maximum portability.

Innovative Integration's products employ the latest digital signal processor technology thereby providing you the competitive edge so critical in today's global markets. Using our powerful data acquisition and DSP products allows you to incorporate leading-edge technology into your system without the risk normally associated with advanced product development. Your efforts are channeled into the area you know best ... your application.

## *Vocabulary*

**What is Malibu?**
Malibu is the Innovative Integration-authored component suite, which combines with the Borland BCB or Microsoft MSVC Integrated Development Environments (IDEs) to support programming of Innovative hardware products. Malibu supports both high-speed data streaming plus asynchronous mailbox communications between the DSP and the Host PC, plus a wealth of Host functions to visualize and post-process data received from or to be sent to the target DSP.

**What is C++ Builder?**
C++ Builder is a general-purpose code-authoring environment suitable for development of Windows applications of any type. Armada extends the Builder IDE through the addition of functional blocks (VCL components) specifically tailored to perform real-time data streaming functions.

**What is Microsoft MSVC?**
MSVC is a general-purpose code-authoring environment suitable for development of Windows applications of any type. Armada extends the MSVC IDE through the addition of dynamically created MSVC-compatible C++ classes specifically tailored to perform real-time data streaming functions.

**What kinds of applications are possible with Innovative Integration hardware?**
Data acquisition, data logging, stimulus-response and signal processing jobs are easily solved with Innovative Integration baseboards using the Malibu software. There are a wide selection of peripheral devices available in the Matador DSP product family, for all types of signals from DC to RF frequency applications, video or audio processing. Additionally, multiple Innovative Integration baseboards can be used for a large channel or mixed requirement systems and data acquisition cards from Innovative can be integrated with Innovative's other DSP or data acquisition baseboards for high-performance signal processing.

**Why do I need to use Malibu with my Baseboard?**
One of the biggest issues in using the personal computer for data collection, control, and communications applications is the relatively poor real-time performance associated with the system. Despite the high computational power of the PC, it cannot reliably respond to real-time events at rates much faster than a few hundred hertz. The PC is really best at processing data, not collecting it. In fact, most modern operating systems like Windows are simply not focused on real-time performance, but rather on ease of use and convenience. Word processing and spreadsheets are simply not high-performance real-time tasks.

The solution to this problem is to provide specialized hardware assistance responsible solely for real- time tasks. Much the same as a dedicated video subsystem is required for adequate display performance, dedicated hardware for real-time data collection and signal processing is needed. This is precisely the focus of our baseboards – a high performance, state-of-the-art, dedicated digital signal processor coupled with real-time data I/O capable of flowing data via a 64-bit PCI bus interface.

The hardware is really only half the story. The other half is the Malibu software tool set which uses state of the art software techniques to bring our baseboards to life in the Windows environment. These software tools allow you to create applications for your baseboard that encompass the whole job - from high speed data acquisition, to the user interface.

**Finding detailed information on Malibu**
Information on Malibu is available in a variety of forms:

- Data Sheet (http://www.innovative-dsp.com/products/malibu.htm)
- On-line Help
- Innovative Integration Technical Support
- Innovative Integration Web Site (www.innovative-dsp.com)

## Online Help

The online help system for Malibu is fully integrated into the excellent OpenHelp system provided with Builder. Help for Malibu is provided in a single file, Malibu.hlp which is installed beneath the main Builder C++ directory tree during the

default installation. It provides detailed information about the components contained in Malibu - their Properties, Methods, Events, and usage examples. An equivalent version of this help file in HTML help format is also provided: Malibu.chm, for use within the MSVC context.

### Innovative Integration Technical Support

Innovative includes a variety of technical support facilities as part of the Malibu toolset. Telephone hotline supported is available via

Hotline (805) 578-4260 8:00AM-5:00 PM PST.

Alternately, you may e-mail your technical questions at any time to:

techsprt@innovative-dsp.com.

### Innovative Integration Web Site

Additional information on Innovative Integration hardware and the Malibu Toolset is available via the Innovative Integration website at www.innovative-dsp.com

## *Typographic Conventions*

This manual uses the typefaces described below to indicate special text.

| Typeface | Meaning |
|---|---|
| `Source Listing` | Text in this style represents text as it appears onscreen or in code. It also represents anything you must type. |
| **Boldface** | Text in this style is used to strongly emphasize certain words. |
| *Emphasis* | Text in this style is used to emphasize certain words, such as new terms. |
| **Cpp Variable** | Text in this style represents C++ variables |
| *Cpp Symbol* | Text in this style represents C++ identifiers, such as class, function, or type names. |
| **KEYCAPS** | Text in this style indicates a key on your keyboard. For example, "Press **ESC** to exit a menu". |
| Menu Command | Text in this style represents menu commands. For example "Click View \| Tools \| Customize" |

# *Windows Installation*

This chapter describes the software and hardware installation procedure for the Windows platform (WindowsXP and Vista).

> **Do *NOT*** install the hardware card into your system at this time. This will follow the software installation.

## *Host Hardware Requirements*

The software development tools require an IBM or 100% compatible Pentium IV - class or higher machine for proper operation. An Intel-brand processor CPU is *strongly recommended*, since AMD and other "clone" processors are not guaranteed to be compatible with the Intel MMX and SIMD instruction-set extensions which the Armada and Malibu Host libraries utilize extensively to improve processing performance within a number of its components. The host system must have at least 128 Mbytes of memory (256MB recommended), 100 Mbytes available hard disk space, and a DVD-ROM drive. Windows2000 or WindowsXP (referred to herein simply as *Windows*) is required to run the developer's package software, and are the target operating systems for which host software development is supported.

## *Software Installation*

The development package installation program will guide you through the installation process.

> **Note:** Before installing the host development libraries (VCL components or MFC classes), you must have Microsoft MSVC Studio (version 9 or later) and/or Codegear RAD Studio C++ (version 11) installed on your system, depending on which of these IDEs you plan to use for Host development. If you are planning on using these environments, it is imperative that they are tested and known-operational before proceeding with the library installation. If these items are not installed prior to running the Innovative Integration install, the installation program will not permit installation of the associated development libraries. However, drivers and DLLs may be installed to facilitate field deployment.

You must have **Administrator Privileges** to install and run the software/hardware onto your system, refer to the Windows documentation for details on how to get these privileges.

To begin the installation, start Windows. Shut down all running programs and disable anti-virus software. Insert the installation **DVD**. If Autostart is enabled on your system, the install program will launch. If the DVD does not Autostart, click on Start | Run... Enter the path to the SETUP.EXE program located at the root of your DVD-ROM drive (i.e. E:\SETUP.EXE) and click "OK" to launch the setup program.

After launching Setup.exe, you will be presented with the following screen.

Using this interface, specify which product to install, and where on your system to install it.

1) Select the appropriate product from the Product Menu.

2) Specify the path where the development package files are to be installed. You may type a path or click "**Change**" to browse for, or create, a directory. If left unchanged, the install will use the default location of "C:\Innovative".

3) Typically, most users will perform a "Full Install" by leaving all items in the "**Components to Install**" box checked. If you do not wish to install a particular item, simply uncheck it. The Installer will alert you and automatically uncheck any item that requires a development environment that is not detected on your system.

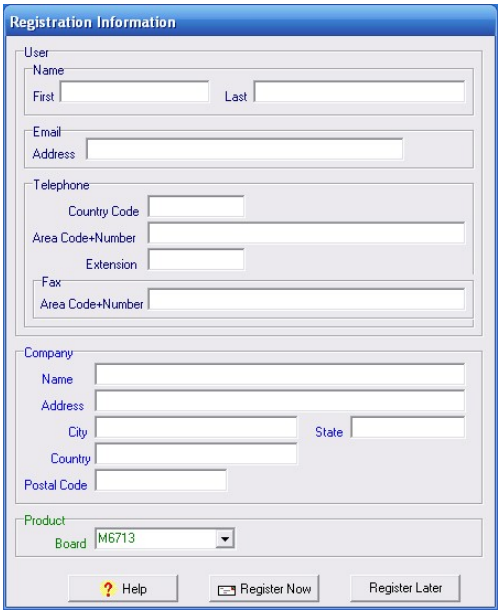
4) Click the Install button to begin the installation.

**Note:** The default “Product Filter” setting for the installer interface is “Current Only” as indicated by the combo box located at the top right of the screen. If the install that you require does not appear in the “Product Selection Box” (1), Change the “Product Filter” to “Current plus Legacy”.

Each item of the checklist in the screen shown above, has a sub-install associated with it and will open a sub-install screen if checked. For example, the first sub-install for “Quadia - Applets, Examples, Docs, and Pismo libraries” is shown below.

The installation will display a progress window, similar to the one shown below, for each item checked.

## *Tools Registration*



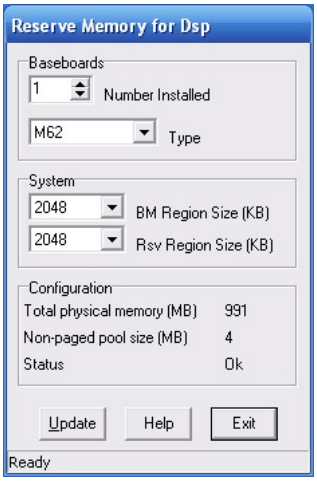

At the end of the installation process you will be prompted to register. If you decide that you would like to register at a later time, click "Register Later".

When you are ready to register, click Start | All Programs | Innovative | <Board Name> | Applets. Open the New User folder and launch **NewUser.exe** to start the registration application. The registration form to the left will be displayed.

Before beginning DSP and Host software development, you must register your installation with Innovative Integration. Technical support will not be provided until registration is successfully completed. Additionally, some development applets will not operate until unlocked with a passcode provided during the registration process.

It is recommend that you completely fill out this form and return it to Innovative Integration, via email or fax. Upon receipt, Innovative Integration will provide access codes to enable technical support and unrestricted access to applets.

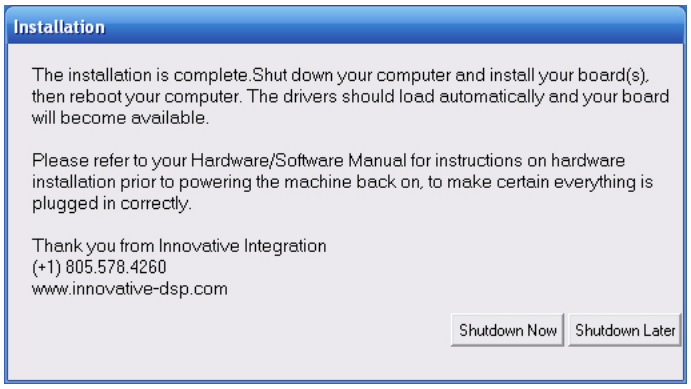
**Bus Master Memory Reservation Applet.**
At the conclusion of the installation process, **ReserveMem.exe** will run (except for SBC products). This will allow you to set the memory size needed for the busmastering to occur properly. This applet may be run from the start menu later if you need to change the parameters.




For optimum performance each Matador Family Baseboard requires 2 MB of memory to be reserved for its use. To reserve this memory, the registry must be updated using the ReserveMem applet. Simply select the **Number of Baseboards** you have on your system, click **Update** and the applet will update the registry for you. If at any time you change the number of boards in your system, then you must invoke this applet found in Start | All Programs | Innovative | <target board> | Applets | Reserve Memory.

After updating the system exit the applet by clicking the **exit** button to resume the installation process.

At the end of the install process, the following screen will appear.


Installation
The installation is complete.Shut down your computer and install your board(s), then reboot your computer. The drivers should load automatically and your board will become available.
Please refer to your Hardware/Software Manual for instructions on hardware installation prior to powering the machine back on, to make certain everything is plugged in correctly.
Thank you from Innovative Integration
(+1) 805.578.4260
www.innovative-dsp.com
Shutdown Now
Shutdown Later

Click the "Shutdown Now" button to shut down your computer. Once the shutdown process is complete unplug the system power cord from the power outlet and proceed to the next section, "Hardware Installation."

## *Hardware Installation*

Now that the software components of the Development Package have been installed the next step is to configure and install your hardware. Detailed instructions on board installation are given in the Hardware Installation chapter, following this chapter.

> **IMPORTANT:** Many of our high speed cards, especially the PMC and XMC Families, require forced air from a fan on the board for cooling. Operating the board without proper airflow may lead to improper functioning, poor results, and even permanent physical damage to the board. These boards also have temperature monitoring features to check the operating temperature. The board may also be designed to intentionally fail on over-temperature to avoid permanent damage. See the specific hardware information for airflow requirements.

## *After Power-up*

After completing the installation, boot your system into Windows.

Innovative Integration boards are plug and play compliant, allowing Windows to detect them and auto-configure at start-up. Under rare circumstances, Windows will fail to auto-install the device-drivers for the JTAG and baseboards. If this happens, please refer to the "TroubleShooting" section.

# *Installation on Linux*

This chapter contains instruction on the installation of the baseboard software for Linux operating systems.

Software installation on Linux is performed by loading a number of **packages**. A Package is a special kind of archive file that contains not only the files that are to be installed, but also installation scripts and dependency information to allow a smooth fit into the system. This information allows the package to be removed, or patched. Innovative uses RPM packages in its installs.

**Package File Names**
A package file name such as Malibu-LinuxPeriphLib-1.1-3.i586.rpm encodes a lot of information.

| **Package Name** | | **Package ID** | | **Information Fields** | |
|---|---|---|---|---|---|
| **Distribution** | **Subpackage** | **Version** | **Revision** | **Hardware Type** | **Extension** |
| Malibu-Linux | PeriphLib | 1.1 | 3 | i586 | .rpm |

## *Prerequisites for Installation*

In order to properly use the baseboard example programs and to develop software using the baseboard, some packages need to be installed before the actual baseboard package.

### The Redistribution Package Group - MalibuRed

This set of packages contain the libraries and drivers needed to run a program using Malibu. This group is called "MalibuRed" because it contains the packages needed to allow running Malibu based programs on a target, non-development machine. (Red is short for 'redistributable').

| **MalibuRed Packages** | **Description** |
|---|---|
| WinDriver-9.2-1.i586.rpm | Installs WinDriver 9.2 release. |
| MalibuLinux-Red-**[ver]**-**[rel]**.i586.rpm | Installs Baseboard Driver Kernel Plugin. |
| intel-ipp_rti-5.3p.x32.rpm | Installs Intel IPP library redistributable files. |

The installation CD, or the web site contains a file called **LinuxNotes.pdf** giving instructions on how to load these packages and how to install the drivers onto your Linux machine. This file is also loaded onto the target machine by the the Malibu-LinuxRed RPM. These procedures need to be completed for every target machine.

### Malibu

To develop software for a baseboard the Malibu packages also must be installed.

| Malibu Packages | Description |
|---|---|
| Malibu-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm | Installs Malibu Source, Libraries and Examples. |

### Other Software

Our examples use the DialogBlocks designer software and wxWidgets GUI library package for user interface code. If you wish to rebuild the example programs you will have to install this software as well.

| Package | Company | URL |
|---|---|---|
| wxWidgets | wxWidgets | http://www.wxwidgets.org |
| DialogBlocks | Anthemion | http://www.anthemion.co.uk.org/dialogblocks |

## *Baseboard Package Installation Procedure*

Each baseboard installation for Linux consists of one or more package files containing self-extracting packages of compressed files, as listed in the table below. Note that package version codes may vary from those listed in the table.

Each of these packages automatically extract files into the `/usr/Innovative` folder, herein referred to as the *Innovative root folder* in the text that follows. For example, the X5-400 RPM extracts into `/usr/Innovative/X5-400-[ver]`. A symbolic link named `X5-400` is then created pointing to the version directory to allow a single name to apply to any version that is in use.

## Board Packages

| Baseboard | Packages | Description |
|---|---|---|
| X5-400M | Malibu-LinuxPeriphLib-[**ver]-[rel**].i586.rpm | Board files and examples. |
| X5-210M | X5-210M-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| X3-10M | X3-10M-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| X3-25M | X3-25M-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| X3-A4D4 | X3-A4D4-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| X3-SD | X3-SD-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| X3-SDF | X3-SDF-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| X3-Servo | X3-Servo-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |
| SBC-ComEx | Sbc-ComEx-LinuxPeriphLib-**[ver]-[rel**].i586.rpm | Board files and examples. |

## Unpacking the Package

As root, type:

```
rpm -i -h X5-400-LinuxPeriphLib-1.1-4.i586.rpm
```

This extracts the X5-400 board files into the Innovative root directory. Use the package for the particular board you are installing.

### Creating Symbolic Links

The example programs assume that the user has created symbolic links for the installed board packages. A script file is provided to simplify this operation by the Malibu Red package. In the MalibuRed/KerPlug directory, there is a script called quicklink.

```
quicklink X5-400 1.1
```

These commands will create a symbolic link `X5-400` pointing to `X5-400-1.1`.

This script can be moved to the user's `bin` directory to allow it to be run from any directory.

### Completing the Board Install

The normal board install is complete with the installation of the files. The board driver install is already complete with the loading of the Malibu Red package. If there are any board-specific steps they will be listed at the end of this chapter.

## *Linux Directory Structure*

When a board package is installed, its files are placed under the `/usr/Innovative` folder. The base directory is named after the board with a version number attached -- for example, the version 2.0 X5-400 RPM extracts into `/usr/Innovative/X5-400-2.0.`

This allows multiple version of installs to coexist by using a symbolic link to point to a particular version. Changing the symbolic link changes with version will be used.

Under the main directory there are a number of subdirectories.

**Applets**
The applets subdirectory contains small application programs that aid in the use of the board. For example, there is a Finder program that allows the user to flash an LED on the board to determine which board is associated with a target number. See the Applets chapter for a fuller description of the applets for a board.

**Documentation**
This directory contains any documentation files for the project. Open the index.html file in the directory with a web browser to see the available files and a description of the contents.

**Examples**
This directory and its subdirectories contain the projects, source and example programs for the board.

**Hardware**
This directory contains files associated with programming the board Logic and any logic images provided.

# *Writing Custom Applications*

The SBC-ComEx carrier card is high performance carrier module which accepts an industry-standard COMEX processor module and up to two XMC I/O modules. All of the I/O features of the COMEX processor module and installed XMC modules are made available via standard connectors for SATA, USB, Ethernet, etc. These are standard peripherals, documented extensively elsewhere, and therefore will not be discussed further in this chapter.

However, there are a number of unique I/O devices on the SBC-ComEx carrier which are not controlled automatically via the operating system or BIOS. Among these peripherals are the onboard PLL, digital I/O ports and optional Tyco A1029 GPS. These devices are mapped as custom resources onto the PCI bus of the COMEX module and may be controlled using features of the Innovative::Sbc-ComEx object within the Malibu libraries as detailed in the following paragraphs.

## *SBC-ComEx Example Software*

The SBC-ComEx `TestbedApp` example in the software distribution, demonstrates functionality of the non-standard carrier hardware features. It consists of a host program executable and source code, which works with the default firmware provided in the board's flash ROM. It is based on the Innovative Malibu software libraries to accomplish low-level device control.

### Tools Required

In general, writing applications for the SBC-ComEx requires the development of host program. This requires a development environment, a debugger, and a set of support libraries from Innovative.

**Table 1. Development Tools for the SBC-ComEx Example**

| Processor | Development Environment | Innovative Toolset | Project Directory |
|---|---|---|---|
| Host PC | Borland Developers Studio C++ | Malibu | Examples\Snap\Bcb11 |
| | Microsoft Visual Studio .NET | | Examples\Snap\VC9 |
| | Anthemion Dialogblocks | | Examples\Snap\DialogBlocks |
| | Common Host Code | | Examples\Snap\Common |

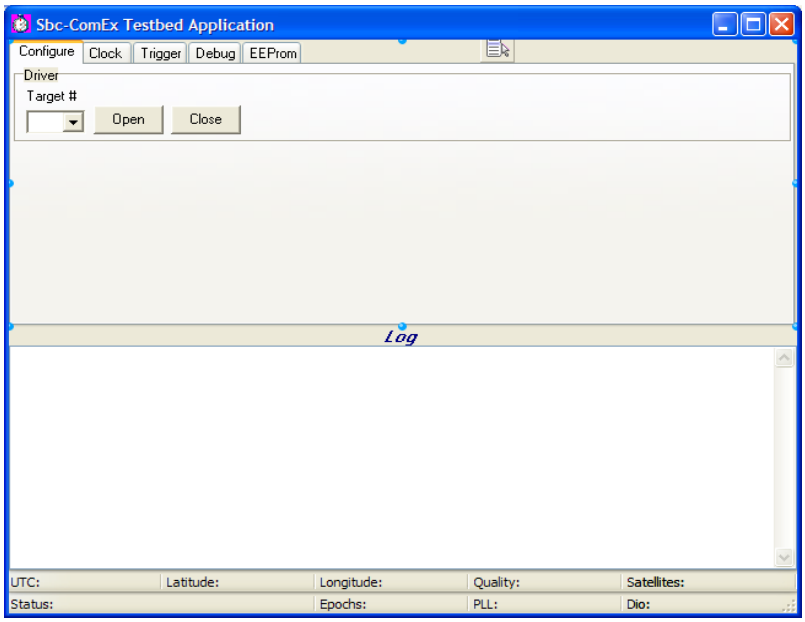
On the host side, the Malibu library is provided in source form, plus pre-compiled Microsoft, Borland or GCC libraries. The application code that implements the entirety of the board-specific functionality of example is factored into the `ApplicationIo.cpp/h` unit. All User Interface aspects of the program are completely independent from the code in `ApplicationIo`, which contains code portable to either compilation environment (i.e., it is common code). While each compiler implements the GUI differently, each version of the example project uses the same file to interact with the hardware and acquire data.

### Program Design

The example is designed to illustrate access to the onboard, low-jitter sample clock and digital I/O ports. Additionally, for carriers equipped with the GPS option, a means of accessing fields parsed from serial records send from the Tyco GPS unit to the Host is shown.

## *The Host Application*

The picture to the right shows the main window of SBC-ComEx example. This form is from the designer of the BCB11 version of the example, but the MSVC version is similar. It shows the layout of the controls of the User Interface.




### User Interface

This application has five tabs. Each tab has its own significance and usage, though few are inter-related. All these tabs share a common Log area, which displays messages and feedback throughout the operation of the program.

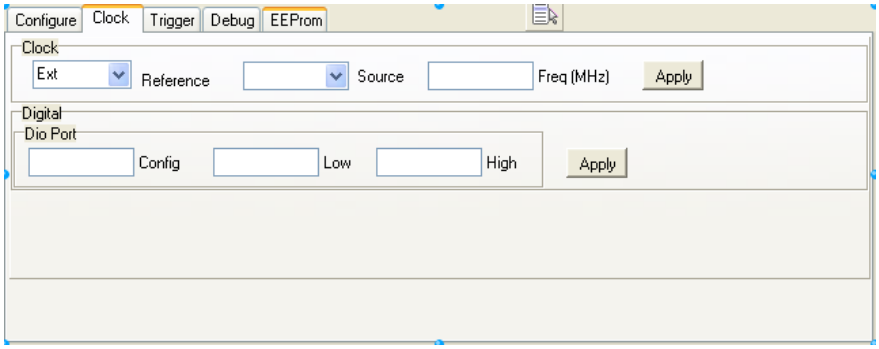
### Configure Tab

As soon as the application is launched, the Configure tab is displayed. In this tab, a combo box is available to allow the selection of the device from those present in the system. All SBC-ComEx devices share a sequence of target number identifiers. The first board found is Target 0, the second Target 1, and so on. This combo box is dynamically filled with available targets detected following a PCI bus scan.

Select an available target, then click the `Open` button to open the driver. To change targets, click the `Close` button to close the driver, select the number of the desired target using the `Target #` combo box, then click `Open` to open communications with the specified target module. The order of the targets is determined by the location of the Sbc-ComEx peripherals on the PCI bus. Since these peripherals are fixed on the SBC-ComEx design, target zero will be used universally under the current software.

**Clock Tab**

This tab has a set of controls that configure the onboard AD9510 PLL which can act as a sample clock for XMC modules installed in sites 0 or 1. Additionally, a second group of controls allows configuration of the digital I/O port pins available on the baseboard.




The controls within the `Clock` group box support configuration and routing of the clock.

The `Clock | Source` combo box specifies whether the XMC modules are clocked via the output of the onboard PLL (setting=`Internal`), or via a user-supplied external signal applied to the EXT CLK IN connector (setting=`External`). If the `Source` is set to `External`, the settings of the `Reference` and `Freq` controls are ignored.

When the `Source` is set to `Internal`, the `Clock | Reference` combo box specifies whether reference clock for the PLL is sourced by a user-supplied signal applied to the EXT CLK IN connector (setting=`Ext`), or by an onboard 10 MHz crystal oscillator (setting=`Crystal`) or by the 10 MHz output the GPS circuit (setting=`Gps`), which is locked to global time. The value of the `Freq (MHz)` edit box specifies the desired PLL output frequency.

Use the `Clock | Apply` button to apply the control settings to the hardware.

The controls in the `Digital | Dio Port` group support configuration of the direction of the digital I/O pins on the carrier.

The value of the `Dio Port | Configure` edit control is treated as a bit-mask. Each bit within the mask controls the direction of a bank of eight DIO bits using the convention that a zero value for that bit configures that bank for input., while a one configures for output. Bit zero in the mask corresponds to the direction of DIO bits 0..7, bit one controls DIO bits 8..15, etc. The mask value is applied to the hardware via the `Sbc_ComEx` module object through the `Dio()` sub-object, via

```
Board.Dio().DioPortConfig(Settings.DioConfig);
```

Following configuration, the state of the DIO port can be read or written at any time. When reading, bits configured as inputs will return the current state of each pin in the bank, whereas pins configured as outputs will return the value last written to the bank. When writing, bits configured as outputs will assume the written state immediately following the write operation, whereas bits configured for input will remain unchanged.

To access to the low-order 32-bits of the DIO port, use the methods shown below.

Reading:

```
int state = Board.Dio().DioPortData().Value();
```

Writing

```
Board.Dio().DioPortData().Value(state);
```

To access to the upper-order 16-bits of the DIO port, use the methods shown below.

Reading:

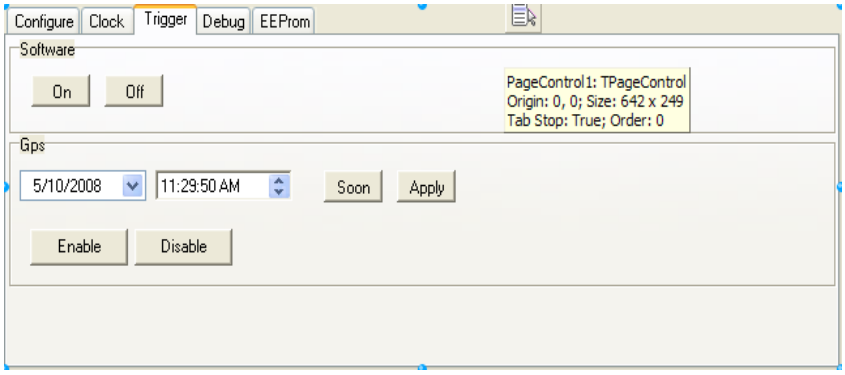
```
short state = Board.Dio().DioPortDataHigh().Value();
```

Writing

```
Board.Dio().DioPortDataHigh().Value(state);
```

**Trigger Tab**

Features on the trigger tab illustrate access to the triggering features of the board.




The trigger signal generated by the carrier FPGA is routed to each of the two XMC sites. The source of the trigger may be either a software command or a user-specified date and time accurate to within one microsecond synchronous with the epoch output from the Tyco GPS module.

To control the software trigger, click `Software | On` or `Software | Off` buttons to enable and disable, respectively.

To program the date and time when the GPS-initiated trigger should fire, adjust the `Gps | Date` and `Gps | Time` controls, then click the `Gps | Apply` button. Alternately, clicking the `Gps | Soon` button sets the time and date five seconds from the current date and time. Finally, click the `Gps | Enable` button to enable the GPS trigger output

**Host Side Program Organization**

The Host example program is designed to be rebuild-able in each of three different host environments: CodeGear RAD Studio and Microsoft Visual Studio using the .NET UI under Windows and DialogBlocks using GCC under Linux. Because Malibu provides a common library within each of these environments, the code that interacts with Malibu is separated out into a class, *ApplicationIo* in the files ApplicationIo.cpp and .h. This class acts identically in all the platforms.

The Main form of the application instantiates an *ApplicationIo* object to perform the work of the example. The UI can call the methods of the *ApplicationIo* to perform the work when, for example, a button is pressed or a control is changed.

Sometimes, however, the *ApplicationIo* object needs to 'call back into' the UI. But since portability is essential, it can't use a pointer to the main window or form, as this would make *ApplicationIo* dependent on the details of Borland, MSVC or DialogBlocks.

The solution used to decouple the *ApplicationIo* from the form is to use an interface class to hide the communications implementation. An interface class is an abstract class that defines a set of methods that can be called by a client (here,

*ApplicationIo*). Within the GUI unit, a concrete version of the interface is constructed by inheriting from the interface. Within the concrete implementation, user interface actions are forwarded UI form class to perform the action. *ApplicationIo* remains completely decoupled from the GUI implementation since it manipulates only a pointer to the abstract class, which is initialized to point to the concrete implementation.

The predefined *IUserInterface* interface class is defined in ApplicationIo.h. The constructor of *ApplicationIo* requires a pointer to the interface, which is saved and used to perform the actual updates to the UI inside of *ApplicationIo's* methods.

## ApplicationIo

### Initialization

The main form creates an ApplicationIo object in its constructor. The object creates a number of Malibu objects at once as can be seen from this detail from the header ApplicationIo.h.

```
// Fields
bool                          FOpened;

// Data
Innovative::Sbc_ComEx         Board;
IUserInterface *              UI;
Innovative::SoftwareTimer     Timer;
unsigned int                  Lost;
unsigned int                  EpochTally;
Innovative::Scripter          Script;
  ...
```

In Malibu, objects are defined to represent units of hardware as well as software units. The `Sbc-ComEx` object represents the COM carrier board. A `Scripter` object can be used to add a simple scripting language to the application, for the purposes of performing hardware initialization during FPGA firmware development. The `SoftwareTimer` object is used to perform operations periodically, within a background thread.

When the GUI is started, an `ApplicationIo` object is instantiated via code substantially similar to the following Borland code snippet:

```
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner), UI(new UserInterface(this)), Io(new ApplicationIo(UI))
{
        SetSettings();

//      OutputClockComboBoxChange(this);
}
```

This constructor creates a concrete instance of the `UserInterface` class, which is passed to a newly-created instance of `ApplicationIo` by pointer, so that the `ApplicationIo` can notify the UI at strategic times during execution of board-specific functions.

The `ApplicationIo` object latches the address of the UI concrete class within a private variable called `UI`, for use in any of its methods or event handlers, as shown below.

```
ApplicationIo::ApplicationIo(IUserInterface * ui)
        : FOpened(false), UI(ui), EpochTally(0)
{
```

```
    Board.OnEpoch.SetEvent(this, &ApplicationIo::HandleOnEpoch);
    OnLog.SetEvent(this, &ApplicationIo::HandleOnLog);
    OnStatus.SetEvent(this, &ApplicationIo::HandleOnStatus);
    Timer.OnElapsed.SetEvent(this, &ApplicationIo::HandleTimer);

    Status(sStatus, "Status: Idle");

    Status(sStatus, "Status: Running");
    Timer.Enabled(true);
}
```

Within the constructor, Malibu software events are linked to callback functions, which are simply methods of the `ApplicationIo` object, via the `SetEvent` method intrinsic to all `OpenWire::Event` objects.

```
// Hook script event handlers.
Board.OnEpoch.SetEvent(this, &ApplicationIo::HandleOnEpoch);
OnLog.SetEvent(this, &ApplicationIo::HandleOnLog);
OnStatus.SetEvent(this, &ApplicationIo::HandleOnStatus);
Timer.OnElapsed.SetEvent(this, &ApplicationIo::HandleTimer);
```

This code attaches event handlers to their corresponding events. Malibu *events* allow functions to be 'plugged into' library classes to be called at certain times or in response to certain events detected. Events allow a tight integration between an application and the library. For example, the `Board.OnEpoch` event fires when a GPS epoch event occurs, once per second. The `ApplicationIo::HandleOnEpoch` method contains code which is executed once per second, when the epoch event fires.

Similarly, `HandleTimer`, handles events issued when the software timer elapses. These handlers could be designed to perform multiple tasks as events occur including displaying messages for user.

```
Timer.OnElapsed.SetEvent(this, &ApplicationIo::HandleTimer);
Timer.OnElapsed.Thunk();
```

In this example, a Malibu SoftwareTimer object has been added to the ApplicationIo class to provide periodic status updates to the user interface. The handler above serves this purpose.

Every event may be are configured to execute in the callers thread context (unsynchronized), or within the main GUI thread context (thunked or synchronized). The latter should be used whenever the handler is designed to perform any sort of user-interface operation, since UI actions are not reentrant and must be executed within the GUI main thread context.

An event may not necessarily be called in the same thread as the main UI thread. If it is not, and if you want to call a UI function in the handler you have to have the event synchronized with the UI thread. A call to `Synchronize()` directs the event to call the event handler in the main UI thread context. This results in a slight performance penalty, but allows us to call UI methods in the event handler freely. The Timer uses a similar synchronization method, `Thunk()`. Here the event is called in the main thread context, but the issuing thread does not wait for the event to be handled before proceeding. This method is useful for notification events.

Creating a hardware object does not attach it to the hardware. The object has to be explicitly opened. The `Open()` method activates the board for use. It opens the device driver and allocates internal resources for use.

```
void ApplicationIo::Open()
{
    if (FOpened)
        return;

    UI->GetSettings();
```

```
    //
    //  Open Devices
    Board.Target(Settings.Target);
    Board.Target(0);    // use target 0 for now
    Board.Open();
    FOpened = true;
    Log("Carrier driver opened...");

    DisplayLogicVersion();

    EpochTally = 0;
}
```

After the driver is opened, we capture and display some information to the screen. This includes the logic version, PCB type and family code. .

```
void  ApplicationIo::DisplayLogicVersion()
{
    {
    std::stringstream msg;
    msg << std::hex << "Logic Revision: " << Board.PciLogicVersion().PciLogicRevision()
        << ", Family: " << Board.PciLogicVersion().PciLogicFamily()
        << ", Pcb: " << Board.PciLogicVersion().PciLogicPcb()
        << ", Type: " << Board.PciLogicVersion().PciLogicType();
    Log(msg.str());
    }
}
```

Similarly, the `Close()` method closes the hardware. Inside this method, first we disable the GPS interrupt handler to avoid spurious ISR handling during the close operation. Then, the module is detached from the hardware and its resources are released.

```
void ApplicationIo::Close()
{
    if (!FOpened)
        return;

    Board.GpsEnabled(false);
    Board.Close();
    FOpened = false;

    Log("Carrier driver closed...");
}
```

**Using the Programmable Timebase**

The carrier includes a low-jitter, programmable PLL which may be used as a timebase (sample clock) for either or both of the installed XMC modules. The `SetClock` method applies the timebase-related settings cached within the Settings object to the PLL hardware, as shown below:

```
//---------------------------------------------------------------------------
//  ApplicationIo::StartClock() --   Enable the clock output using current settings
//---------------------------------------------------------------------------

void ApplicationIo::SetClock()
{
```

The call to the UI `GetSettings` method is used to refresh the Settings cache from the controls on the UI form.

```
        UI->GetSettings();
```

The PLL timebase may either generate a sample clock, or it can simply steer a clock connected to the EXT CLOCK connector to the XMC sites, controlled via the `Board.ClockSource` method. If the PLL is generating the clock, its frequency is specified via the `Board.Clock.Frequency` method and the source for the PLL reference clock is programmed via the `Board.ReferenceSource` method.

```
        // Disable outputs during frequency changes
        Board.ReferenceSource(static_cast<Sbc_ComEx::IIRefSource>(Settings.PllReferenceSource));
        Board.ClockSource(static_cast<Sbc_ComEx::IIClockSource>(Settings.OutputClock));
        Board.Clock().Frequency(Settings.OutputFrequency*1.e6);
```

The onboard PLL allows generation of a high-performance sample clock over a wide range of frequencies. However, due to limitation of the onboard VCO, the actual output frequency may not precisely match the requested frequency. The `Board.Clock().FrequencyActual()` method can be used to retrieve the actual clock frequency, as shown above.

```
        {
        std::stringstream msg;
        msg << "PLL actual frequency: " << std::scientific << std::setprecision(9) <<
            Board.Clock().FrequencyActual();
        Log(msg.str());
        }
```

**Using Programmable Bit I/O**

The carrier features forty-eight bits of programmable bit I/O. The direction of bit I/O may be programmed in groups of eight bits, via the Board.Dio().DioPortConfig method. The parameter of this method is a bit mask, in which each mask bit corresponds to eight I/O bits - Bit zero controls port bits 0..7, bit 1 controls port bits 8..15, etc. Bit groups configured for output will change state when written and will return their current programmed state when read. Bit groups configured for input will not change when written and will return the current input state when read.

Bits 0..31 of the available I/O bits written or read via the `Board.Dio().DioPortData()` property methods. Bits 32..47 of the available I/O bits written or read via the `Board.Dio().DioPortDataHigh()` property methods, as shown below.

```
//-----------------------------------------------------------------------------
//  ApplicationIo::SetDio() --
//-----------------------------------------------------------------------------

void ApplicationIo::SetDio()
{
        UI->GetSettings();

    Board.Dio().DioPortConfig(Settings.DioConfig);
    Board.Dio().DioPortData().Value(Settings.DioDataLow);
    Board.Dio().DioPortDataHigh().Value(Settings.DioDataHigh);
}
```

**Polling Thread**

The ApplicationIo object for the Sbc-ComEx employs a background thread to implement polling operations which is used to update the user interface periodically with the PLL lock status and the current readback state of the digital I/O bits. The

polling operation is implemented through use of an `Innovative::SoftwareTimer` object whose sole event handler, `OnElapsed`, is configured to call the `ApplicationIo::HandleTimer` method, illustrated below.

```
//---------------------------------------------------------------------------
//  ApplicationIo::HandleTimer() --  Periodic status check
//---------------------------------------------------------------------------

void ApplicationIo::HandleTimer(OpenWire::NotifyEvent & event)
{
    if (!FOpened)
        return;

    {
        std::stringstream msg;
        msg << "PLL: " << (Board.Clock().Locked() ? "Locked" : "Unlocked");
        Status(sPll, msg.str());
    }
    {
        std::stringstream msg;
        msg << "Dio: " << std::hex << Board.Dio().DioPortDataHigh().Field(0, 16)
         << " " << Board.Dio().DioPortData().Value();
        Status(sDio, msg.str());
    }
}
```

Calls to the `ApplicationIo::Status` method forward the specified text string through the IUserInterface object pointer, `UI`, into the main form, as shown below.

```
//---------------------------------------------------------------------------
//  ApplicationIo::Status() --  Log message thunked to main thread
//---------------------------------------------------------------------------

void  ApplicationIo::Status(IIStatusType type, const std::string & msg)
{
    StatusMessageEvent e(type, msg);
    OnStatus.Execute(e);
}
//---------------------------------------------------------------------------
//  ApplicationIo::HandleOnStatus() --
//---------------------------------------------------------------------------

void ApplicationIo::HandleOnStatus(StatusMessageEvent & Event)
{
    UI->Status(Event.Type, Event.Message);
}
```

Inspection of the prototype for `Innovative::SoftwareTimer` within SoftwareTimer_Mb.h shows that the `OnElapsed` event is of type `OpenWire::ThunkedEventHandler`, which means that the call `ApplicationIo::HandleOnStatus` will automatically be thunked into the foreground thread context at runtime, making it safe to perform UI updates within the called UI method.

### GPS Support

The Sbc-ComEx carrier supports an optional Tyco GPS plug-in module for precision timebase synchronization and position tracking. If installed, the module may be enabled via the `GpsEnabled` method, after which the `IsGpsAntenna` and `IsGpsLocked` methods may be used to check whether a GPS antenna is properly installed and satellite locking has occurred, respectively.

Features of the Tyco A1029 GPS module may be accessed via the `Gps()` sub-object of the `Innovative::Sbc_ComEx`. Code within the `Innovative::Sbc_ComEx` object performs processing of GPS messages using a internal interrupt service routine (ISR) which executes at epoch (one-second) intervals.

By default, the GPS module is programmed to emit the standard GPS sentences listed below. These messages are automatically parsed within the built-in ISR and the results of the parsing operation is stored in the structures listed below:

| Structure | GPS Sentence | Description |
|---|---|---|
| `Time()` | Zda | UTC Time / Date and Local Time Zone Offset |
| `Course()` | Vtg | Course Over Ground and Ground Speed |
| `FixData()` | Gga | Global Positioning System Fix Data |
| `RmsData()` | Rms | Recommended Minimum Specific GPS Data |
| `Satellites()` | Gsa | GPS DOP and Active Satellites |
| `View()` | Gsv | GPS Satellites in View |

See the *Tyco GPS Firmware A1029 / A1035-C User's Manual* for details on these standard GPS sentences.

The A1029 supports a large number of additional messages which may be enabled by sending commands to the the device via the TycoGps::Command method. If additional sentences are enabled in this fashion, they will ignored during ISR processing but can be analyzed within your application code by installing an Sbc_ComEx::OnEpoch event handler, as illustrated below.

```
//-------------------------------------------------------------------------
//  ApplicationIo::HandleOnEpoch() --  Periodic status check
//-------------------------------------------------------------------------

void ApplicationIo::HandleOnEpoch(Innovative::EpochEvent & Event)
{
    StringList & list(Event.List);
    for (StringList::iterator i = list.begin(); i != list.end(); ++i)
        Log(*i);

       std::stringstream msg;
       msg << "Epochs: " << ++EpochTally;

    Status(sEpoch, msg.str());

    {
    std::stringstream msg;
```

```
        msg << "UTC: " << Board.Gps().Time().Utc;
        Status(sUtc, msg.str());
        }
        {
        std::stringstream msg;
        msg << "Latitude: " << Board.Gps().FixData().Latitude.Value
            << " " << Board.Gps().FixData().Latitude.Units;
        Status(sLatitude, msg.str());
        }
        {
        std::stringstream msg;
        msg << "Longitude: " << Board.Gps().FixData().Longitude.Value
            << " " << Board.Gps().FixData().Longitude.Units;
        Status(sLongitude, msg.str());
        }
        {
        std::stringstream msg;
        msg << "Quality: " << Board.Gps().FixData().FixQuality;
        Status(sQuality, msg.str());
        }
        {
        std::stringstream msg;
        msg << "Satellites: " << Board.Gps().FixData().SatellitesInUse;
        Status(sSatellites, msg.str());
        }
    }
```

This event handler inspects the state of the parsed `Gps` sentence structures and provides a mechanism for accessing unparsed messages. The parameter to this event handler is an object of type `Innovative::EpochEvent`, which is merely a wrapper on a `Innovative::StringList` containing all unparsed sentences emitted by the A1029 , stored as a collection of `std::strings`.

```
    //============================================================================
    // CLASS EpochEvent  --
    //============================================================================

    class EpochEvent : public OpenWire::Event
    {
    public:
        StringList List;

        EpochEvent(StringList & list)
            :  List(list)
            {
            }
    }
```

The `Sbc_ComEx::GpsTriggerTime` method is used to specify a future date and time when the trigger hardware on the Sbc-ComEx is to be armed. Within the ISR, the current time reported by the GPS is compared to the time specified by this method. When they match , the trigger is armed. The hardware will automatically activate the trigger at the inception of the next epoch event, accurate to within 1 uS, allowing multiple Sbc-ComEx modules located throughout the world to initiate I/O simultaneously within 1 uS of one-another.

# *Developing Host Applications*

Developing an application will more than likely involve using an integrated development environment (IDE) , also known as an integrated design environment or an integrated debugging environment. This is a type of computer software that assists computer programmers in developing software.

The following sections will aid in the initial set-up of these applications in describing what needs to be set in Project Options or Project Properties.

## *Borland Turbo C++*

**BCB10 (Borland Turbo C++) Project Settings**

When creating a new application with File, New, VCL Forms Application - C++ Builder

**Change the Project Options for the Compiler:**

Project Options
++ Compiler (bcc32)
C++ Compatibility
**Check 'zero-length empty base class (-Ve)'**
**Check 'zero-length empty class member functions (-Vx)'**

In our example Host Applications, if not checked an access violation will occur when attempting to enter any event function.

i.e.
Access Violation OnLoadMsg.Execute – Load Message Event
Because of statement
Board->OnLoadMsg.SetEvent( this, &ApplicationIo::DoLoadMsg );

**Change the Project Options for the Linker:**

Project Options
Linker (ilink32)
Linking – **uncheck 'Use Dynamic RTL'**

In our example Host Applications, if not unchecked, this will cause the execution to fail before the Form is constructed.

Error: First chance exception at $xxxxxxxx. Exception class EAccessViolation with message "Access Violation!"
Process ???.exe (nnnn)

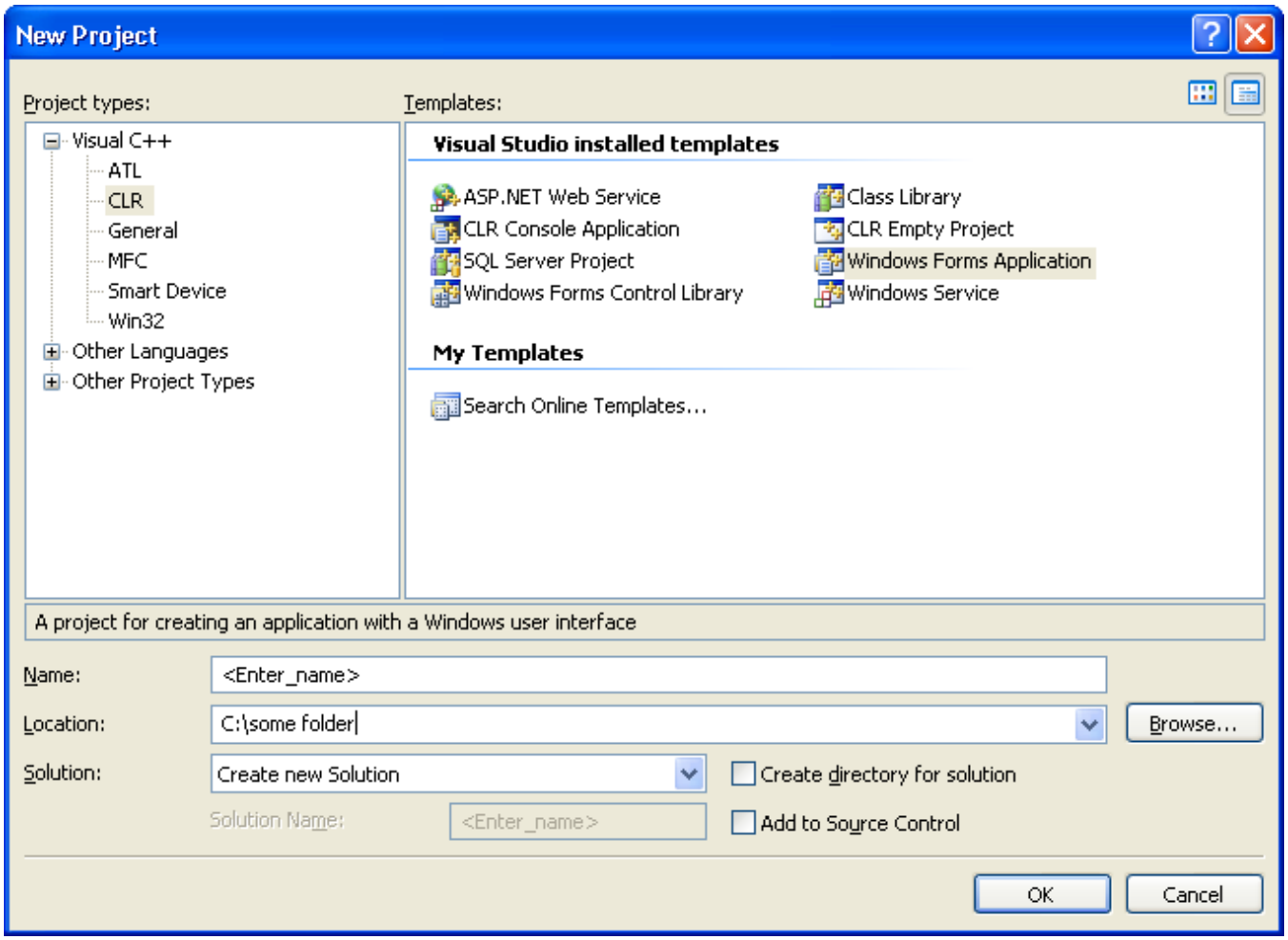
**Other considerations:**

Project Options
++ Compiler (bcc32)
Output Settings
**check – Specify output directory for object files(-n)**
**(release build) Release**
**(debug build) Debug**
Paths and Defines
**add Malibu**
Pre-compiled headers
**uncheck everything**
Linker (ilink32)
Output Settings
**check – Final output directory**
**(release build) Release**
**(debug build) Debug**
Paths and Defines
(ensure that Build Configuration is set to All Configurations)
**add Lib/Bcb10**
(change Build Configuration to Release Build)
**add lib\bcb10\release**
(change Build Configuration to Debug Build)
**add lib\bcb10\debug**
(change Build Configuration back to All Configurations)

Packages
**uncheck - Build with runtime packages**

**Microsoft Visual C++ 2005 (version 8) Project Properties**

When creating a new application with File, New, Project with Widows Forms Application:



*Project Properties (Alt+F7)*
*Configuration Properties*

| Project Defaults | |
|---|---|
| Configuration Type | Application (.exe) |
| Use of MFC | Use Standard Windows Libraries |
| Use of ATL | Not Using ATL |
| Minimize CRT Use in ATL | No |
| Character Set | Use Unicode Character Set |
| Common Language Runtime support | Common Language Runtime Support (/clr) |
| Whole Program Optimization | No Whole Program Optimization |

*C++*
*General*
*Additional Include Directories*

*Malibu*
*PlotLab/Include – for graph/scope display*

*Code Generation*
*Run Time Library*

*Multi-threaded Debug DLL (/Mdd)*

*Precompiled Headers*
*Create/Use Precompile Headers*

*Not Using Precompiled Headers*

*Linker*
*Additional Library Directories*

*Innovative\Lib\Vc8*

If anything appears to be missing, view any of the example sample code Vc8 projects.

## *Summary*

Developing Host and target applications utilizing Innovative DSP products is straightforward when armed with the appropriate development tools and information.

# *Applets*

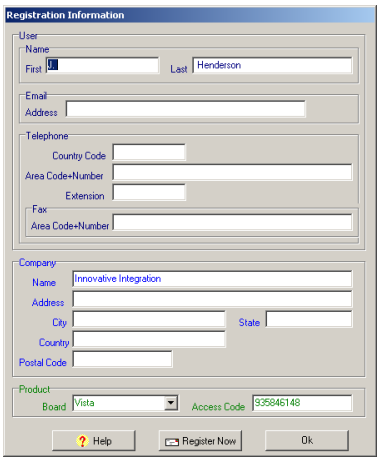
The software release for a baseboard contains programs in addition to the example projects. These are collectively called "applets". They provide a variety of services ranging from post analysis of acquired data to loading programs and logic to a full replacement host user interface. The applets provided with this release are described in this chapter.

Shortcuts to these utilities are installed in Windows by the installation. To invoke any of these utilities, go to the Start Menu | Programs | <<Baseboard Name>> and double-click the shortcut for the program you are interested in running.

## *Common Applets*

### Registration Utility (NewUser.exe)

Some of the Host applets provided in the Developers Package are keyed to allow Innovative to obtain end-user contact information. These utilities allow unrestricted use during a 20 day trial period, after which you are required to register your package with Innovative. After, the trial period operation will be disallowed until the unlock code provided as part of the registration is entered into the applet. After using the NewUser.exe applet to provide Innovative Integration with your registration information, you will receive:



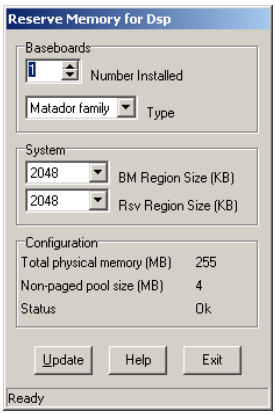

The unlock code necessary for unrestricted use of the Host applets

A WSC (tech-support service code) enabling free software maintenance downloads of development kit software and telephone technical hot line support for a one year period.

### Reserve Memory Applet (ReserveMemDsp.exe)

Each Innovative PCI-based DSP baseboard requires 2 to 8 MB of memory to be reserved for its use, depending on the rates of bus-master transfer traffic which each baseboard will generate. Applications operating at transfer rates in excess of 20 MB/sec should reserve additional, contiguous busmaster memory to ensure gap-free data acquisition.

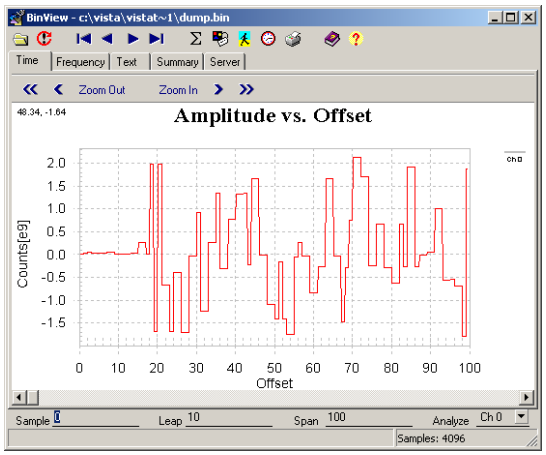
To reserve this memory, the registry must be updated using the ReserveMemDsp applet. If at any time you change the number of or rearrange the baseboards in your system, then you must invoke this applet to reserve the proper space for the busmaster region. See the Help file ReserveMemDsp.hlp, for operational details.




## *Data Analysis Applets*

### Binary File Viewer Utility (BinView.exe)

BinView is a data display tool specifically designed to allow simplified viewing of binary data stored in data files or a resident in shared DSP memory. Please see the on-line BinView help file in your Binview installation directory.



# *Carrier Hardware*

## *Introduction*

Sbc-ComEx is a single board computer host card integrating a COM Express computer module, dual XMC module sites and on-card peripherals for IO support and control.

The COM Express CPU module provides standard PC architecture built around Intel processors and chipsets. Features include USB, Ethernet, PCI Express, video, audio and SATA HDD. The XMC modules use PCI Express to from the COM Express CPU module. The card boots from HDD, USB Flash drive, or Ethernet.

On card peripherals feature rate generation support IO on the XMC sites and can be slaved to a GPS receiver. An FPGA for custom IO requirements is connected to the local PCI bus of the COM Express CPU module.

Software is Windows or Linux, and standard PC software is supported.

### Features

**Supports COM Express CPU Modules**

- Type 2 (minimum resources PCIe x5, GBE x1, SATA x2, PEG x16, VGA)
- Multiple vendors – Radisys, Kontron, Advantech, Adlink
- Core2 Duo, Celeron
- Runs Windows and Linux

**Dual XMC Sites**

- Supports X3, X5 families from Innovative
- Supports VITA 42.3 modules
- Site 0 = 4x lanes
- Site 1 = 8x lanes (uses PEG lanes 0-7)

**Host Card Peripherals**

- Ethernet port 10/100/1000
- USB 2.0 (minimum configuration is 4 external, one internal)
- Solid State Disk drive site
- GPS Receiver Module
- Sample rate generation using PLL with VCO and post-divider

**PCI Interface**

32 bit, 33 MHz interface, 3V

Slave Only, no busmastering

interrupt for GPS interface

FPGA

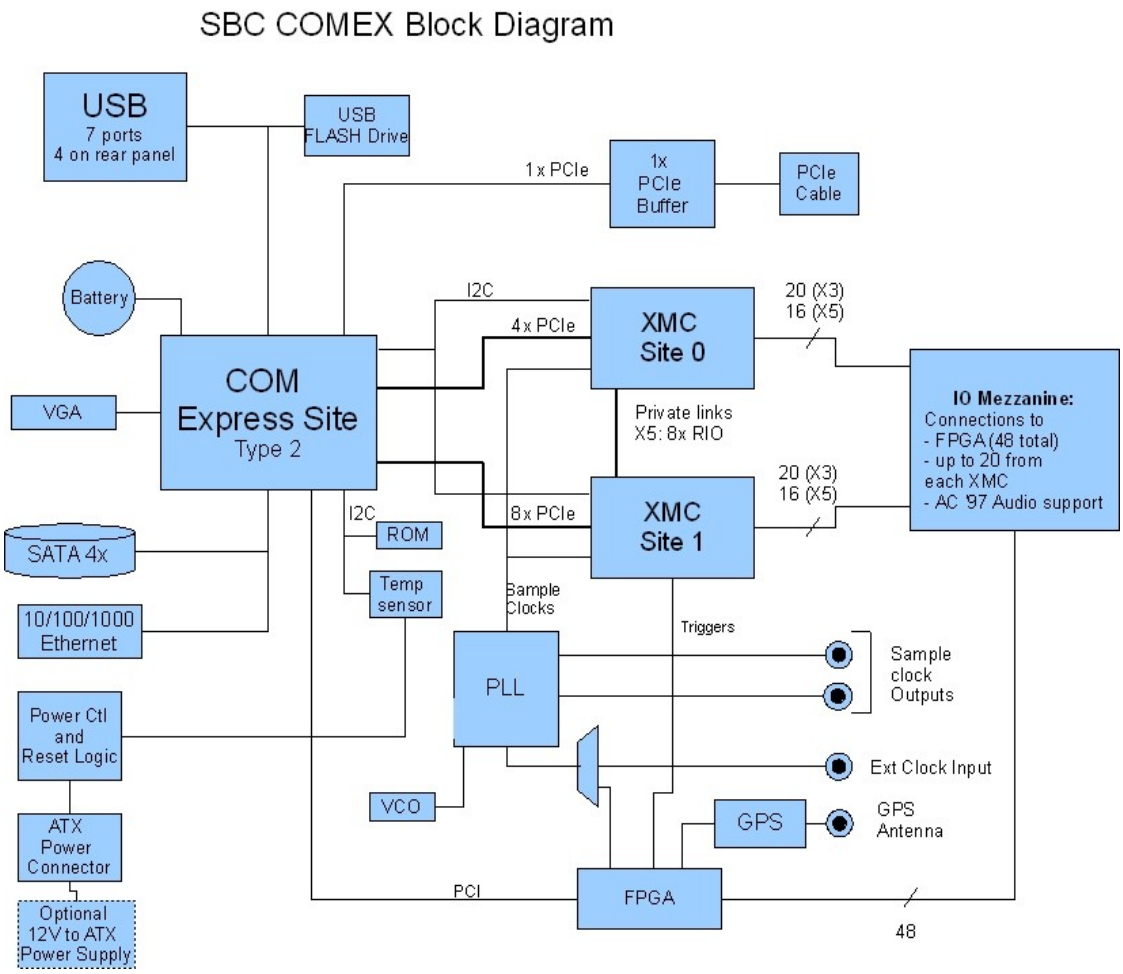
- Xilinx Spartan 3E 250K gate
- Performs PCI interface and control
- Configures from on-card Xilinx XCF04 FLASH
- FLASH is reprogrammable using XSVF JTAG method (same as PMCs)

~250 mm x 170 mm three card stack

Product Codes and Variants

80199-0 SBC COMEX

Block Diagram



FPGA for IO

PCI bus Interface

Resource requirements

SBC COMEX FPGA requires 1M bytes of memory space (into which peripherals are mapped) and no interrupts. There is a single base address register in memory space for SBC COMEX FPGA. These resources are requested as part of the Plug-n-Play boot operation and are not programmable.

Memory map

PCI into mem address space (BAR0)

| PCI Memory WORD Address (Base + ) | PCI Memory Address Base + (hex) | Logic Addr. (hex) | Read/ Write | Description |
|---|---|---|---|---|
| 0x0.. 0x12400 | 0x00000.. 0x49000 | 0x00-0x49 | - | - |
| 0x12800 | 0x4a000 | 0x4a | R/W | IDROM serial flash interface [idrom_wr, idrom_rd] |
| 0x12c00 | 0x4b000 | 0x4b | R/W | FPA (FPGA Config Flash Programming) |
| 0x13000 | 0x4c000 | 0x4c | | - |
| 0x14400 | 0x51000 | 0x51 | W | PCI Interrupt Mask Register: |
| 0x14800 | 0x52000 | 0x52 | R | Temperature |
| 0x14C00 | 0x53000 | 0x53 | R/W | Temperature Warning (W) / Clear (R) |
| 0x15000 | 0x540xx | 0x54 | R/W | Temperature Fail (W) / Clear ( R ) |
| 0x15400 | 0x550xx | 0x55 | | - |
| 0x15800 | 0x56000 | 0x56 | | - |
| 0x15c00 | 0x57000 | 0x57 | R | PCI Revision |
| 0x16000 | 0x58000 | 0x58 | | - |
| 0x16400 | 0x59000 | 0x59 | | - |
| 0x16800 | 0x5a000 | 0x5a | | - |
| 0x16c00 | 0x5b000 | 0x5b | | - |
| 0x17000 | 0x5c000 | 0x5c | W | PCI Test Reg |

| | | | | |
|---|---|---|---|---|
| 0x17400 | 0x5d000 | 0x5d | W | Software Triggers |
| 0x17800 | 0x5e000 | 0x5e | | - |
| 0x17c00 | 0x5f000 | 0x5f | | - |
| 0x19000 | 0x64000 | 0x64 | R | PCI Interrupt status read: (clears Interrupts on read) |
| 0x19400 | 0x65000 | 0x65 | W | PCI Control |
| 0x19800 | 0x66000 | 0x66 | R | PCI Status |
| 0x19C00 | 0x67000 | 0x67 | | - |
| 0x1A000 | 0x68000 | 0x68 | R/W | GPS Serial Port |
| 0x1A400 | 0x69000 | 0x69 | R/W | GPS Control/status |
| 0x1A800 | 0x6A000 | 0x6A | R/W | Triggering Controls |
| 0x1AC00 | 0x6B000 | 0x6B | R/W | DIO[31:0] |
| 0x1B000 | 0x6C000 | 0x6C | R/W | DIO[47:32] |
| 0x1B400 | 0x6D000 | 0x6D | R/W | DIO Enables |
| 0x1B800 | 0x6E000 | 0x6E | R/W | GPS UART control/status |
| 0x1BC00 | 0x6F000 | 0x6F | R/W | Decimation |
| 0x1C000 | 0x70000 | 0x70 | R/W | GPS Trigger Enable |
| 0x1C400 | 0x71000 | 0x71 | R/W | PLL Interface (AD9511) |
| 0x1C800 | 0x72000 | 0x72 | R/W | Clock control register |
| 0x1Cc00 | 0x73000 | 0x73 | | - |

Slave mode operation

SBC COMEX supports slave mode operation for configuration. Slave accesses never burst to the card.

All slave mode accesses are set to “medium” decode. This means that SBC COMEX has at least one cycle at the beginning of every access as a wait state.

## Bus mastering Operations

None are supported.

Registers

**Calibration Serial Flash Register** (0x4a000, read, write)

This register is similar to the ones used on Omnibus modules and X3 modules. The data and clock must be controlled using a similar bit-bang software interface, to emulate the I2C interface.

| Bit | Definition |
|---|---|
| 0 | SDA serial data bit (read/write) |
| 1 | SCK bit for serial clock (write only) |
| 31..2 | Unused |

### *Revision REGISTER (0x57000, read only)*

This register reports the logic and hardware revisions.

| Bit | Function |
|---|---|
| 15..0 | Logic rev code (set in logic) |
| 19..16 | Hardware family |
| 23..20 | Hardware revision code (RevA, B, etc.) |
| 27..24 | Hardware type |
| 31..28 | Unused – read as 0 |

## ***DEVICE CONTROL REGISTER*** *(0x65000)*

| Bit | Function |
|---|---|
| 0 | Not used |
| 1 | LED signal** '1' = off. |
| 31..2 | - |
| 15..10 | Software Triggers |
| 31..16 | - |

** If the board LED fails to work, the board either did not enumerate properly

FLASH JTAG Interface (FPA) (0x4B000)

The FLASH JTAG interface allows host software to modify the contents of the Xilinx XCF04 device used to configure the PCI control logic.

The FLASH JTAG is controlled via a single register in the PCI register set. The following diagram gives the definition of the FLASH JTAG control register.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|
| 0 | JTAG_TMS | O | TMS control |
| 1 | JTAG Output Enable | Internal to FPGA | Output enable control for TCK, TMS, TDI; '0' = tristate (default) |
| 2 | Not used | | |
| 3 | JTAG_CLK | O | TCK control |
| 4 | JTAG_TDI | O | Data input to FLASH |
| 5 | JTAG_TDO | I | Data output from FLASH config ROM |
| 7..6 | - | | 00 |
| 15..7 | | | 0x00 |
| 31..16 | - | | 0x1303 |

The configuration ROM must have a valid image prior to reprogramming for this interface to work. If not, a Xilinx JTAG cable may be used for reprogramming.

The output enable bit must be set to '1' to enable the JTAG signals output. This prevents collision with the Xilinx cable if it is begin used. The bits default to tri-state after power-up.

Temperature Monitoring Interface and Powerdown Behavior

Temperature monitoring for the module consists of a Texas Instruments TMP175 temperature sensor mounted near the center of the module and a logic component that monitors the temperature continuously.

Current temperature can always be read from the temp sensor register. The temperature is updated at approximately a 100 Hz rate with a nominal resolution of 0.06 degrees C. The temperature sensor should be accurate to about 2 degrees C. Keep in mind that the actual temperature at the chips is normally higher by 5 C or so.

The logic component provides a programmable temperature warning and fail level. Both temperature warning and failure are latched when they occur and must be cleared by a read their respective registers.

| Alarm Setting | Temperature ( Celsius) | Set Register to .... |
|---|---|---|
| Warning | 60 | X”3C0” |
| Fail | 65 | X”410” |

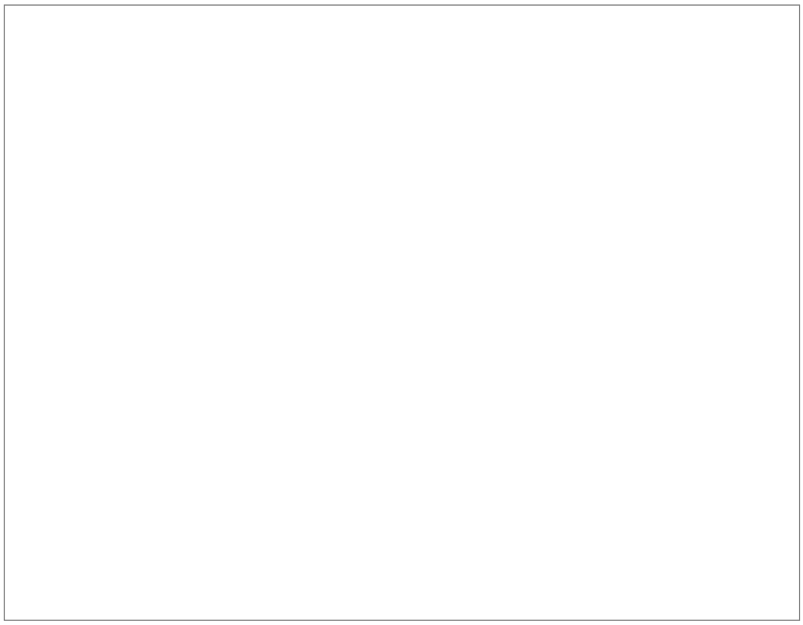
A temperature failure results in a power down signal to the power control electronics to shut down. The SBC COMEX will power down to standby state when a temperature failure occurs.

The temperature sensor must be present and responding for the SBC COMEX to operate. If the temp sensor fails, this is treated as a temperature failure. The logic continues to attempt to communicate with the temperature sensor. If multiple failure conditions are found, the logic should be reloaded.

Temperature (0x52 R)

This register gives the current temperature when read. Scaling from the temperature sensor is a 2's complement number that is sign extended to 16-bits from the 12-bit value given in this table. Note that 0 degrees is 0 output and scale factor is 0.0625 C/bit.

Temperature(C) = reading * 0.0625



| Bits | Function |
|---|---|
| 15..0 | Temperature (read only) |
| (Others) | Not used |

Temperature Warning (0x53 R/W)

This register sets the temperature warning level for writes and clears temperature warning when read.

The warning level should be provided as a temperature reading set point (see table above).

| Bits | Function |
|---|---|
| 15..0 | Temperature |
| 29..16 | - |

| | |
|---|---|
| 30 | Temperature Warning |
| 31 | Temperature Failure |

Temperature Failure (0x54 R/W)

This register sets the temperature failure level for writes and clears temperature failure when read.

The failure level should be provided as a temperature reading set point (see table above).

| Bits | Function |
|---|---|
| 15..0 | Temperature |
| 29..16 | - |
| 30 | Temperature Warning |
| 31 | Temperature Failure |

## ***PLL Interface*** *(0x71000)*

This register controls PLL interface. Reads and writes to this register location access the AD951x register set. See AD951x data sheet for details. Accesses to this register must be paced by 1ms between accesses.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|
| 23..0 | PLL Data | I/O | PLL instruction/data word. |
| 30..24 | - | | Not Used. |
| 31 | PLL Status | I | PLL status bit (programmable function in AD9511) |

### *PLL Reference Clock Control Register ( 0x72000)*

This register controls the PLL reference selection.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|
| 1..0 | PLL REF Select | O | X0 = PLL reference from FPGA (default) |
| | | | 10 = 100MHz from oscillator |
| | | | 11 = GPS slave 1 MHz |
| 2 | PLL REF enable | 0 | 0 = disabled |
| | | | 1 = enabled(default) |
| 31..3 | - | - | Not Used. |

### *PCI Test Register (0x5D000)*

This register is just for debug and test. No defined use at this time.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|
| 31..0 | PLL Test | I/O | Read/Write test register |

### *DIO 31..0 Register (0x6B000)*

This register is the data for the digital IO.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|
| 31..0 | DIO[31:0] | I/O | Digital IO to mezzanine |

### *DIO 47..32 Register (0x6C000)*

This register is the data for the digital IO.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|
| 15..0 | DIO[47:32] | I/O | Digital IO to mezzanine |
| 31..16 | - | - | - |

### *DIO Enables Register (0x6D000)*

This register is the digital enables bits.

| Bit | Bit Field Name | Direction from PCI FPGA | Function |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 5..0 | DIO_EN | O | Enable Digital IO to mezzanine<br>0= input (default)<br>bit 0 for byte 0, bit 1 for byte 1 .... |

GPS Interface

A GPS module can be used on the SBC COMEX to provide time and position information. A pulse per second input to the FPGA is used to synchronize the PLL reference to GPS time.

The GPS unit has a serial interface for control and status. The protocol for the port is given in the Tyco A1029 GPS manual. The FPGA interface provides serial port interface to the module, status and control. The host configures the GPS to a baud rate of 4800 for default use. The UART has a 16 byte FIFO on transmit (TX) and 2K FIFO on receive (RX). The GPS_TX_RDY bit shows when the TX FIFO has room for at least 4 more bytes. The GPS_RX_NOT_EMPTY shows when the RX FIFO is not empty.

An interrupt can be used to signal the host when GPS data is available, once every second. The interrupt is triggered by the PPS signal from the GPS unit. The host should acknowledge the interrupt by reading the interrupt status register, then read the GPS FIFO for all data.

## ***GPS Serial Port*** *(0x68000)*

| Bit | Direction | Function |
|---|---|---|
| 7..0 | R/W | Serial data;<br>R = RX FIFO read<br>W = TX FIFO write |

| | |
|---|---|
| 31..8 | - |

### *UART Config/Status (0x6E000)*

This register sets provides UART status and control.

| Bit | Bit Field Name | Direction | Function |
|---|---|---|---|
| 2..0 | BAUD Rate | R/W | UART BAUD rate (default = 4800)<br>“000” =115200<br>“001” =57600<br>“010” =38400<br>“011” =19200<br>“100” =9600<br>“101” =4800 |
| 3 | GPS TX RDY | R | TX FIFO ready; '1' = can accept 4 more points<br>'0' = FIFO is almost full |
| 4 | GPS RX NOT EMPTY | R | RX FIFO not empty; '1' = FIFO is not empty |
| 15..5 | GPS_RX_FIFO_RD_COUNT | R | RX FIFO read count -1 (NOT EMPTY is true when only 1 point is available) |
| 31..16 | - | | Not Used. |

### *GPS Status and Control (0x69000)*

| Bit | Direction | Function |
|---|---|---|

| | | |
|---|---|---|
| 0 | R/W | GPS enable. '0' = disabled (default)<br><br>This powers down the unit. GPS will be forced to acquire a new fix. |
| 1 | R | GPS Lock '1' = locked |
| 2 | R | GPS antenna status '1' = good |
| 3 | R/W | GPS FIFO Reset, 1 = reset (default)<br><br>Resets both RX and TX FIFOs. |

***PCI Interrupt Register*** *(0x64000, read)*

This register reports the PCI interrupt status. The status is cleared on read.

| Bit | Function |
|---|---|
| 0 | GPS packet ready |
| 31..1 | - |

***PCI Interrupt Mask Register*** *(0x51000, write)*

This register provides an interrupt mask.

| Bit | Interrupt Source |
|---|---|
| 0 | GPS packet interrupt 0= disabled (default) |
| 31..1 | |

PLL Configuration

The PLL device is an Analog Devices AD9511. The programming of this device is identical to otherAD951x devices used on X3 and ClockGen products.

| Product | Clocking Device | Output | Type | Use | Ref Clock | VCO |
|---|---|---|---|---|---|---|
| SBC COMEX | AD9511 PLL | 0 | LVPECL | FPGA | 100 MHz | 100-140 MHz |
| | | 1 | LVPECL | Clock Out 0 | | |
| | | 2 | LVPECL | Clock Out 1 | | |
| | | 3 | LVDS | XMC 0 | | |
| | | 4 | LVDS | XMC 1 | | |

Triggering Controls

The triggering module provides 6 trigger outputs: two to each module, and two outputs to the mezzanine connector JN1. The triggering modes are software, next GPS epoch, and external input.

### *Software Triggering Control (0x5D000)*

| Bit | Function |
|---|---|
| 0 | Software trigger |
| 31..1 | - |

### *Triggering Control (0x6A000)*

This register controls the triggering features.

| Bit | Bit Field Name | Function |
|---|---|---|
| 23..0 | Frame Count | How many sample clocks to count in frame mode |
| 27..24 | - | |
| 28 | Trigger_run | Enable triggering module; 0=disabled (default) |
| 29 | - | - |
| 30 | Trigger Mode | 0= unframed; 1 = framed |
| 31 | External Trigger Enable | 0 = disabled (default) |

### *Decimation Control (0x6F000)*

This register controls the decimation features.

| Bit | Bit Field Name | Function |
|---|---|---|
| 11..0 | Decimation | Decimate 1/N clocks for each trigger |
| 31..12 | - | Not used |

### *GPS Triggering Control (0x70000)*

This register enables triggering on next GPS PPS rising edge.  Any writes to this register trigger on next  rising edge of PPS.

| Bit | Bit Field Name | Function |
|---|---|---|
| 31.. 0 | GPS Trigger Enable | x |